Introduction to Git

Eric Daoud

March 26, 2019

Content

History

Definition

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Its current maintainer since 2005 is Junio Hamano.

Installation

If you are using Linux, you can install git using the standard package managers, as explained here.
On macos, git is installed with the XCode command line tools. More informations here.

Configuration

- Configure git with your name and email so that we know who authored the commits.

```
git config --global user.name "John Doe"
git config --global user.email "john@example.com"
```

- If you are using Github or alike, export your public ssh key and load it in the settings, so that you can clone repositories with SSH instead of HTTPS.

Your first repository

- Clone an existing directory:

```
git clone git://github.com/user/repo.git
```

- Or create an empty one:

```
mkdir lab-git && cd lab-git
git init
git remote add origin git://github.com/user/repo.git
```

Here, "origin" is the name of the remote repository. You can add several remote repositories, named differently.

Outline

Git architecture

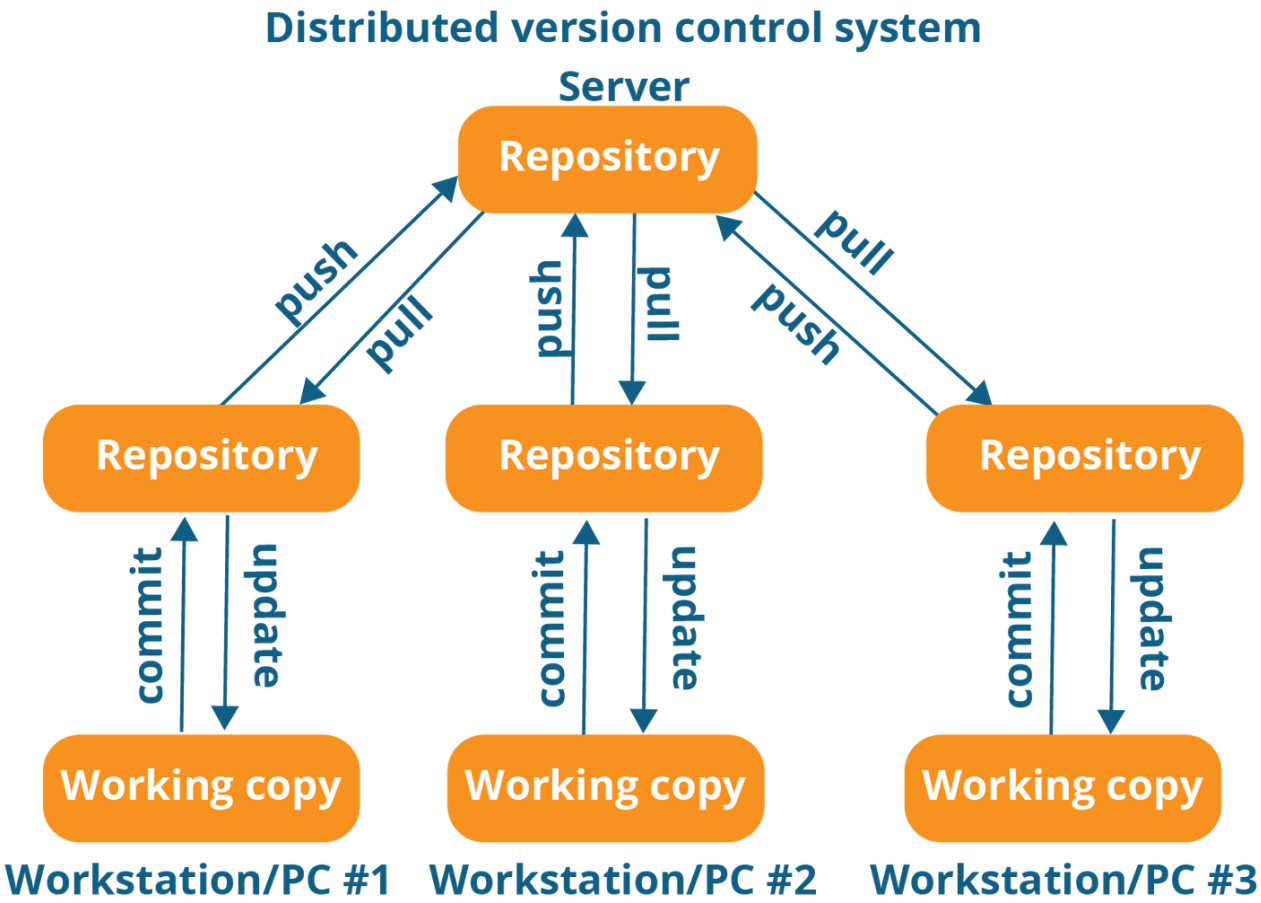

Figure: Git Architecture

Git stages

Stages in local:

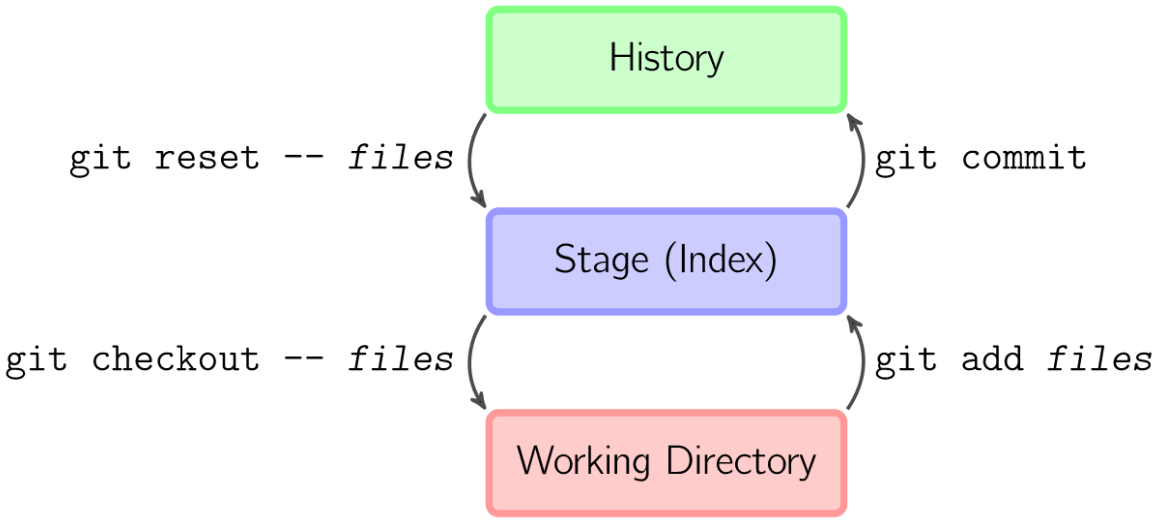

Figure: Git stages

Outline

Tracking files

Files are untracked by default. With the **add** command we tell git we want to stage certain files (for commiting them later).

- **git add file.py** : add a single file
- **git add *.py** : add all python files
- **git add -u** : among the tracked files, add the ones that were modified
- **git add .** : add all files from the current directory
- **git add *** : add all files
- **git status** : check which files are stagged

Once files are stagged, we can either **commit** the changes or discard the changes with **checkout**.

Commit and push

Once we have stagged files, we are ready for a **commit**:

```
git add *.py && git commit -m "adding all python files"
```

After a commit, the changes are stored in local only (run **git log** to see the local commits). We need to push the commits to the distant repository with **push**. Here, we tell git to push our commits to the remote repository (called origin) on the master branch.

```
git push origin master
```

As mentioned earlier, we can push to various remote repositories:

```
git remote add github git://github.com/user/repo.git
git remote add gitlab git://gitlab.com/user/repo.git
git push github master && git push gitlab master
```

Pulling changes

We **pull** from the remote repository when we want to get the latest updates to our working copy.

- **git pull** : pull all branches (fetch + merge)
- **git pull origin master** : pull master branch
- **git pull origin some-feature** : pull *some-feature* branch
- **git fetch** : update the remote tracking branches, without merging

Stash

Sometimes you may have conflicts when pulling other branches into your branch, or git might tell you that the file you are editing changed and it can't pull right now. To save your local changes without commiting anything, use **stash**:

```
git pull origin awesome-feature # fails because of
    conflict_file.py
git stash conflict_file.py # stash changes
git pull origin awesome-feature # works but your changes are
    temporarily gone
git stash apply # re-apply your changes
git commit -m "made the world a better place"
git push origin awesome-feature # now the world really is a
    better place
```

Undo changes

When things went wrong, here's what to do:

- Discard changes on an untracked file: **git checkout file.py**
- Discard changes on all untracked files: **git checkout ***
- Untrack a file (after adding it): **git reset --HEAD file.py**
- Delete last commit but keep the changes: **git reset --soft HEAD~1**
- Delete last commit and delete the changes: **git reset --hard HEAD~1**
- Delete all changes and get your working copy like the remote master branch: **git reset --hard origin/master**
- Delete tracked files from disk and repo: **git rm go_away.py**
- Delete tracked files from repo but keep on disk: **git rm --cached go_away.py**

Ignore Files

Git is meant to track code, not data. You should probably not track all the files in a repository. Typically you want to avoid tracking:

- Large data files, like JSON or CSV
- Credentials
- Compiled files
- etc ...

A good way for avoiding that is using a **.gitignore** file, placed at the root of the repository. In this file, we specify which files we want to avoid tracking, and it might look like this:

```
*.pyc  # ignore all pyc files
*.csv  # ignore all csv files
data/  # ignore the whole data folder
```

Outline

Branches

Use branches to work on separate copies of the production code (master branch), add your features and then merge your branch back into the production branch.

- **git checkout -b branchname** : create a branch and switch to it
- **git checkout master** : switch back to master
- **git merge branchname** : merge the changes from *branchname* into the current branch
- **git push origin branchname** : push *branchname* to the remote repository
- **git branch -d branchname** : delete the local branch *branchname*
- **git push origin :branchname** : delete the remote branch *branchname*

Steps for updating a branch

Make sure you stay up to date with master: merge master to your local branch regularly to avoid conflicts.

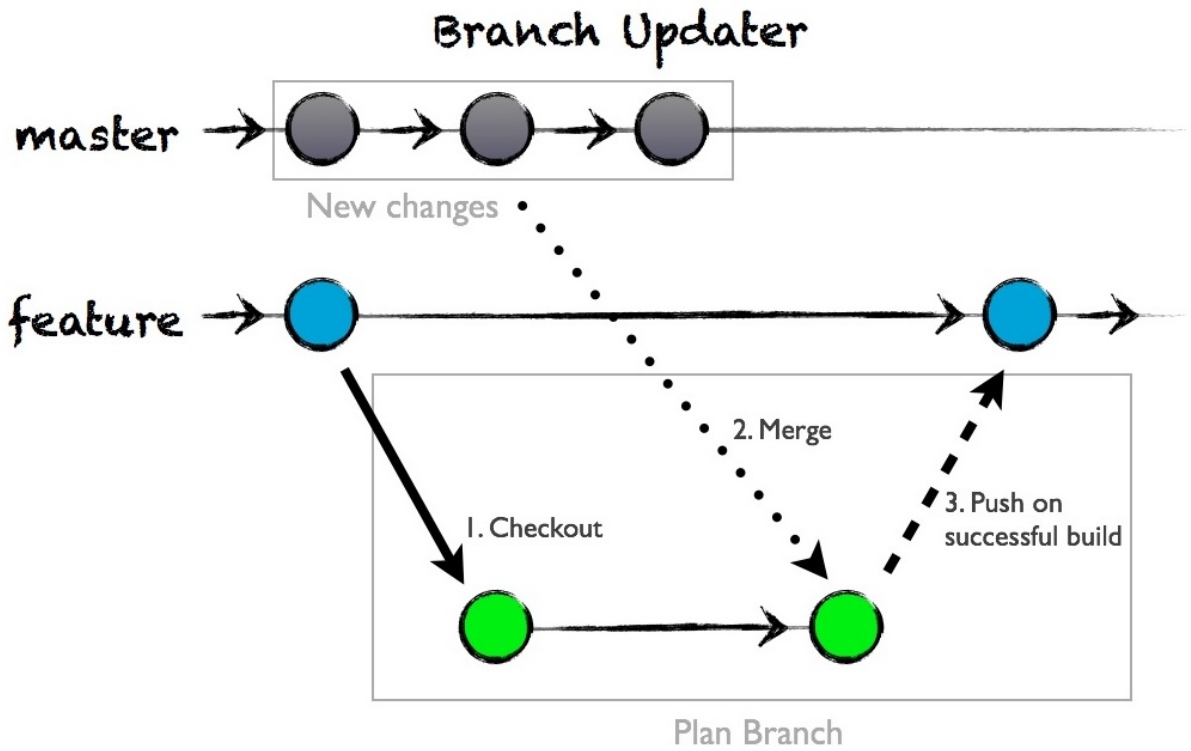

Figure: Updating a branch